JJGO / dstest

Last active last month • Report abuse

Code Revisions 5 Stars 22 Forks 4 Embed <script src="https://(Download ZIP

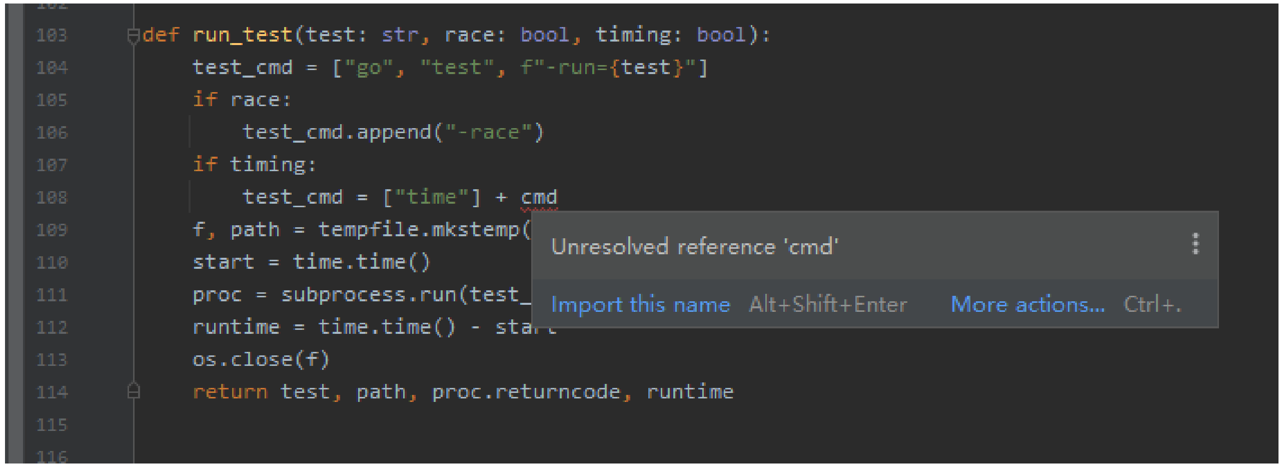
Utility for running MIT 6.824 lab test in parallel and saving failed logs

dstest

```
#!/usr/bin/env python

import itertools
import math
import signal
import subprocess
import tempfile
import shutil
import time
import os
import sys
import datetime
from collections import defaultdict
from concurrent.futures import ThreadPoolExecutor, wait, FIRST_COMPLETED
from dataclasses import dataclass
from pathlib import Path
from typing import List, Optional, Dict, DefaultDict, Tuple

import typer
import rich
from rich import print
from rich.table import Table
from rich.progress import (
    Progress,
    TimeElapsedColumn,
    TimeRemainingColumn,
    TextColumn,
    BarColumn,
    SpinnerColumn,
)
from rich.live import Live
from rich.panel import Panel
from rich.traceback import install

install(show_locals=True)


@dataclass
class StatsMeter:
    """
    Auxiliary classs to keep track of online stats including: count, mean, variance
    Uses Welford's algorithm to compute sample mean and sample variance incrementally.
    https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance#On-line_algorithm
    """

    n: int = 0
    mean: float = 0.0
    S: float = 0.0

    def add(self, datum):
        self.n += 1
        delta = datum - self.mean
        # Mk = Mk-1+ (xk – Mk-1)/k
        self.mean += delta / self.n
        # Sk = Sk-1 + (xk – Mk-1)*(xk – Mk).
        self.S += delta * (datum - self.mean)

    @property
    def variance(self):
        return self.S / self.n

    @property
    def std(self):
        return math.sqrt(self.variance)


def print_results(results: Dict[str, Dict[str, StatsMeter]], timing=False):
```

```python
    table = Table(show_header=True, header_style="bold")
    table.add_column("Test")
    table.add_column("Failed", justify="right")
    table.add_column("Total", justify="right")
    if not timing:
        table.add_column("Time", justify="right")
    else:
        table.add_column("Real Time", justify="right")
        table.add_column("User Time", justify="right")
        table.add_column("System Time", justify="right")

    for test, stats in results.items():
        if stats["completed"].n == 0:
            continue
        color = "green" if stats["failed"].n == 0 else "red"
        row = [
            f"[{color}]{test}[/{color}]",
            str(stats["failed"].n),
            str(stats["completed"].n),
        ]
        if not timing:
            row.append(f"{stats['time'].mean:.2f} ± {stats['time'].std:.2f}")
        else:
            row.extend(
                [
                    f"{stats['real_time'].mean:.2f} ± {stats['real_time'].std:.2f}",
                    f"{stats['user_time'].mean:.2f} ± {stats['user_time'].std:.2f}",
                    f"{stats['system_time'].mean:.2f} ± {stats['system_time'].std:.2f}",
                ]
            )
        table.add_row(*row)

    print(table)


def run_test(test: str, race: bool, timing: bool):
    test_cmd = ["go", "test", f"-run={test}"]
    if race:
        test_cmd.append("-race")
    if timing:
        test_cmd = ["time"] + cmd
    f, path = tempfile.mkstemp()
    start = time.time()
    proc = subprocess.run(test_cmd, stdout=f, stderr=f)
    runtime = time.time() - start
    os.close(f)
    return test, path, proc.returncode, runtime


def last_line(file: str) -> str:
    with open(file, "rb") as f:
        f.seek(-2, os.SEEK_END)
        while f.read(1) != b"\n":
            f.seek(-2, os.SEEK_CUR)
        line = f.readline().decode()
    return line


# fmt: off
def run_tests(
    tests: List[str],
    sequential: bool       = typer.Option(False,  '--sequential',      '-s',    help='Run all test of each group in order'),
    workers: int           = typer.Option(1,      '--workers',         '-p',    help='Number of parallel tasks'),
    iterations: int        = typer.Option(10,     '--iter',            '-n',    help='Number of iterations to run'),
    output: Optional[Path] = typer.Option(None,   '--output',          '-o',    help='Output path to use'),
    verbose: int           = typer.Option(0,      '--verbose',         '-v',    help='Verbosity level', count=True),
    archive: bool          = typer.Option(False,  '--archive',         '-a',    help='Save all logs intead of only failed ones'),
    race: bool             = typer.Option(False,  '--race/--no-race',  '-r/-R', help='Run with race checker'),
    loop: bool             = typer.Option(False,  '--loop',            '-l',    help='Run continuously'),
    growth: int            = typer.Option(10,     '--growth',          '-g',    help='Growth ratio of iterations when using --loop'),
    timing: bool           = typer.Option(False,   '--timing',          '-t',    help='Report timing, only works on macOS'),
    # fmt: on
):

    if output is None:
        timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
        output = Path(timestamp)
    if race:
        print("[yellow]Running with the race detector\n[/yellow]")

    if verbose > 0:
        print(f"[yellow] Verbosity level set to {verbose}[/yellow]")
        os.environ['VERBOSE'] = str(verbose)

    while True:

        total = iterations * len(tests)
        completed = 0

        results = {test: defaultdict(StatsMeter) for test in tests}

        if sequential:
            test_instances = itertools.chain.from_iterable(itertools.repeat(test, iterations) for test in tests)
        else:
            test_instances = itertools.chain.from_iterable(itertools.repeat(tests, iterations))
        test_instances = iter(test_instances)

        total_progress = Progress(
            "[progress.description]{task.description}",
            BarColumn(),
            TimeRemainingColumn(),
            "[progress.percentage]{task.percentage:>3.0f}%",
            TimeElapsedColumn(),
        )
        total_task = total_progress.add_task("[yellow]Tests[/yellow]", total=total)

        task_progress = Progress(
            "[progress.description]{task.description}",
            SpinnerColumn(),
            BarColumn(),
            "{task.completed}/{task.total}",
        )
        tasks = {test: task_progress.add_task(test, total=iterations) for test in tests}

        progress_table = Table.grid()
        progress_table.add_row(total_progress)
        progress_table.add_row(Panel.fit(task_progress))

        with Live(progress_table, transient=True) as live:

            def handler(_, frame):
                live.stop()
                print('\n')
                print_results(results)
                sys.exit(1)

            signal.signal(signal.SIGINT, handler)

            with ThreadPoolExecutor(max_workers=workers) as executor:

                futures = []
                while completed < total:
                    n = len(futures)
                    if n < workers:
                        for test in itertools.islice(test_instances, workers-n):
                            futures.append(executor.submit(run_test, test, race, timing))

                    done, not_done = wait(futures, return_when=FIRST_COMPLETED)

                    for future in done:
                        test, path, rc, runtime = future.result()

                        results[test]['completed'].add(1)
                        results[test]['time'].add(runtime)
                        task_progress.update(tasks[test], advance=1)
                        dest = (output / f"{test}_{completed}.log").as_posix()
                        if rc != 0:
                            print(f"Failed test {test} - {dest}")
                            task_progress.update(tasks[test], description=f"[red]{test}[/red]")
                            results[test]['failed'].add(1)
                        else:
                            if results[test]['completed'].n == iterations and results[test]['failed'].n == 0:
                                task_progress.update(tasks[test], description=f"[green]{test}[/green]")

                        if rc != 0 or archive:
                            output.mkdir(exist_ok=True, parents=True)
                            shutil.copy(path, dest)

                        if timing:
                            line = last_line(path)
                            real, _, user, _, system, _ = line.replace(' '*8, '').split(' ')
                            results[test]['real_time'].add(float(real))
                            results[test]['user_time'].add(float(user))
                            results[test]['system_time'].add(float(system))

                        os.remove(path)

                        completed += 1
                        total_progress.update(total_task, advance=1)

                        futures = list(not_done)

        print_results(results, timing)

        if loop:
            iterations *= growth
            print(f"[yellow]Increasing iterations to {iterations}[/yellow]")
        else:
            break


if __name__ == "__main__":
    typer.run(run_tests)
```

iwanttobepowerful commented on Mar 19, 2021



```python
def run_test(test: str, race: bool, timing: bool):
    test_cmd = ["go", "test", f"-run={test}"]
    if race:
        test_cmd.append("-race")
    if timing:
        test_cmd = ["time"] + cmd
    f, path = tempfile.mkstemp(
    start = time.time()
    proc = subprocess.run(test_
    runtime = time.time() - sta
    os.close(f)
    return test, path, proc.returncode, runtime
```

JJGO commented on Mar 19, 2021 Author

@iwanttobepowerful Good catch, fixed

JJGO commented on Mar 22, 2021 Author

I don't see why not

xinyu-zheng commented on Apr 5, 2021

Hi Jose, I think you uploaded dslogs here

JJGO commented on Apr 5, 2021 Author

@timzhengxy Yup, I copied the wrong file when making a revision. Thanks for the heads up.

ruiqurm commented on Apr 25, 2022

How to run all tests separately like in your blog? I only thought of this hard way here:

```
cat test_test.go| grep Test | sed 's\(\ \g'|awk '/func/ {printf "%s ",$2;}' | xargs ./dstest.py -p 4 -o .run -v 1 -r  -s
```

JJGO commented on Apr 26, 2022 Author

Your way is totally workable. There are only a few labs, so you can just keep the strings around in the terminal history and subselect tests as needed (e.g. if they fail more).

That said, I did it in a couple of ways :

- `grep 'func Test'`, paste into vim and wrangle a bit.
- If you want a "more efficient" way, you can collapse every thing preceeding `xargs` with `rg 'func (Test.*)\(' -oNr '$1' test_test.go`

PS: `cat FILE | grep PATTERN` can always be written as `grep PATTERN FILE`

ruiqurm commented on Apr 26, 2022

> Your way is totally workable. There are only a few labs, so you can just keep the strings around in the terminal history and subselect tests as needed (e.g. if they fail more).
>
> That said, I did it in a couple of ways :
>
> - `grep 'func Test'`, paste into vim and wrangle a bit.
> - If you want a "more efficient" way, you can collapse every thing preceeding `xargs` with `rg 'func (Test.*)\(' -oNr '$1' test_test.go`
>
> PS: `cat FILE | grep PATTERN` can always be written as `grep PATTERN FILE`

OK. Thank you. Your testing script helps me a lot.

niebayes commented on Dec 17, 2022

Line 108 `test_cmd = ["time"] + cmd` shall be `test_cmd = ["time"] + test_cmd`.